Stohastičko optimiranje hiperparametara za modelsko prediktivno upravljanje u autonomnim utrkama

Vitez, Gabriela

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 74

STOCHASTIC HYPERPARAMETER OPTIMIZATION FOR MODEL PREDICTIVE CONTROL IN AUTONOMOUS RACING

Gabriela Vitez

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 74

STOCHASTIC HYPERPARAMETER OPTIMIZATION FOR MODEL PREDICTIVE CONTROL IN AUTONOMOUS RACING

Gabriela Vitez

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 04 March 2024

MASTER THESIS ASSIGNMENT No. 74

Student:	**Gabriela Vitez (0036521662)**
Study:	Information and Communication Technology
Profile:	Control Systems and Robotics
Mentor:	prof. Ivan Marković

Title: **Stochastic hyperparameter optimization for model predictive control in autonomous racing**

Description:

Model predictive control for autonomous racing is an optimization problem that is based on solving a nonlinear quadratic problem. The optimization problem is set with the goal of tracking the reference trajectory for a known dynamic model of an autonomous vehicle, which takes into account the presence of certain forces. The quality of the optimal solution depends on a number of hyperparameters of the optimization problem, such as the weights of the criterion function of the state (velocities, vehicle turn rate) and control variables (wheel torques). In order to ensure the finding of autonomous vehicle trajectories with the shortest possible execution time, in most cases the desired hyperparameters are determined manually, which requires expert knowledge and a large amount of time. The goal of this thesis is to implement a stochastic hyperparameter optimization method for model predictive control in autonomous races, which will ensure the automatic determination of hyperparameters that lead to fastest feasible vehicle trajectories. The obtained hyperparameters will be evaluated in a simulation and, if possible, on a real autonomous race car with a comparison of the quality of the obtained trajectories for manually and automatically determined hyperparameters.

Submission date: 28 June 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 74

Pristupnica:	**Gabriela Vitez (0036521662)**
Studij:	Informacijska i komunikacijska tehnologija
Profil:	Automatika i robotika
Mentor:	prof. dr. sc. Ivan Marković

Zadatak: **Stohastičko optimiranje hiperparametara za modelsko prediktivno upravljanje u autonomnim utrkama**

Opis zadatka:

Modelsko prediktivno upravljanje za autonomne utrke optimizacijski je problem koji je zasnovan na rješavanju nelinearnog kvadratnog problema. Optimizacijski problem postavlja se s ciljem praćenja referentne trajektorije za poznati dinamički model autonomnog vozila, koji u obzir uzima prisutnost određenih sila. Kvaliteta optimalnog rješenja ovisi o brojnim hiperparametrima optimizacijskog problema, kao što su težine kriterijske funkcije stanja (brzine, skretanje, zakret vozila) i upravljačkih varijabli (momenti na kotačima). Kako bi se osigurao pronalazak trajektorija autonomnog vozila s najkraćim mogućim vremenom izvođenja, u većini se slučajeva željeni hiperparametri određuju ručno, što zahtijeva ekspertno znanje i veliku količinu vremena. Cilj je ovoga diplomskog rada implementirati stohastičku metodu optimizacije hiperparametara za modelsko prediktivno upravljanje u autonomnim utrkama, koja će osigurati automatsko određivanje hiperparametara s kojima se postižu najbrže izvedive trajektorije vozila. Dobiveni će hiperparametri biti evaluirani u simulaciji te po mogućnosti na stvarnom autonomnom bolidu uz usporedbu kvalitete dobivenih trajektorija za ručno i automatski određene hiperparametre.

Rok za predaju rada: 28. lipnja 2024.

Contents

1 Introduction

Autonomous vehicles are nowadays becoming more relevant in automotive and technology industries, with applications ranging from everyday transportation to high-speed racing. These autonomous systems, including self-driving cars, drones, robotic automation, and other intelligent systems, require precision and efficiency to operate without human intervention. They rely on advanced sensors, machine learning algorithms, and sophisticated control strategies for safe navigation and task execution. In particular, self-driving cars could potentially introduce a lot of benefits: reducing traffic accidents caused by human error, optimizing traffic flow, providing mobility solutions for those unable to drive, and transforming goods transportation. Among them, autonomous racing presents a demanding challenge, because it requires the vehicle to make split-second decisions, while maintaining optimal trajectories. This introduces the need to use sophisticated control algorithms.

Model Predictive Control (MPC) is a popular control strategy in autonomous racing due to its ability to predict future vehicle states and optimize control inputs. Optimization plays a crucial role in the effectiveness of autonomous systems, especially in the context of MPCs. The goal is to identify the parameters that yield the best solution from many possibilities. In autonomous racing, the goal is to minimize lap times while ensuring the vehicle remains stable and on track, requiring careful management of speed and control. This is where hyperparameter tuning comes in.

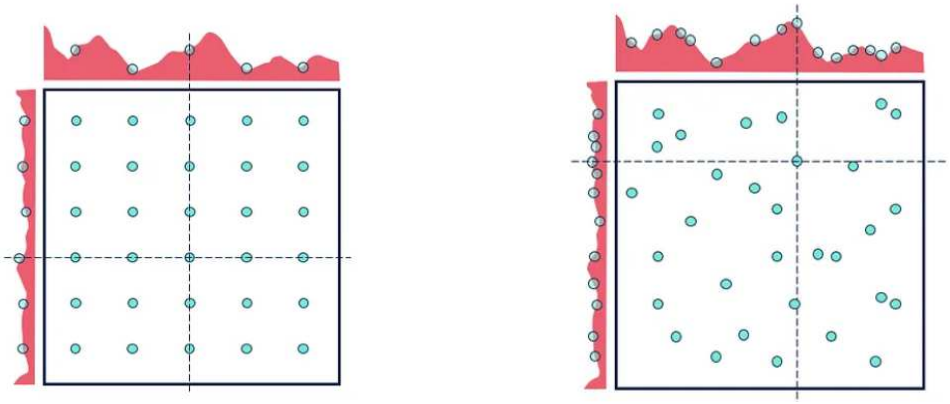
This thesis aims to make the tuning of MPC hyperparameters easier and more efficient by using stochastic optimization methods implemented in MATLAB and Simulink. Stochastic optimization involves techniques that can explore complex hyperparameter spaces to automatically find the optimal ones.

2 Problem Description

Predictive and control models are often used in various fields such as healthcare, finance and facility management. For example, Morovat et al. [1] explored model-based control methods to optimize the timing of preheating in school buildings. Youssef et al. [2] proposed an MPC strategy for optimizing variable speed wind energy conversion systems, demonstrating strong performance under varying wind speeds. In [3], Schwab et al. used predictive models to anticipate which patients might test positive for SARS-CoV-2 or require hospital care.

The efficacy of these models and control systems depends on their design choices – the parameters and hyperparameters. Model parameters, such as neural network weights, are learned during training. On the contrary, hyperparameters (e.g., number of layers, learning rates, activation functions) must be set before training begins. These hyperparameters significantly influence both the learning process and the final performance of the model.

However, tuning hyperparameters is a complex and time-consuming task, often requiring lots of expertise. The sheer number of potential combinations makes manual search for the optimal set impractical. Basic automated hyperparameter search techniques include grid and random search, but these have limited performance. Grid search evaluates each combination within a predefined grid of hyperparameter values, while random search samples combinations randomly (Fig. 2.1). Both methods require significant computational resources and time and they do not use previous experiment results as prior knowledge. Therefore, they become computationally infeasible as the number of hyperparameters increases and they may not guarantee identification of the best hyperparameter set.



(a) Grid Search **(b)** Random Search

Figure 2.1: Traditional automated hyperparameter search methods, source from [4]

Similarly, in MPC systems, hyperparameters, such as prediction and control horizons, state weights, and input rate constraints, need to be tuned beforehand. Traditional approaches like trial-and-error or heuristics require numerous closed-loop simulations and the dependencies between the hyperparameters are non-intuitive.

2.1 Model Predictive Control

MPC is a method which uses a mathematical model of the process to predict future system behaviour and optimize control inputs. A prediction model describes the dynamical relationship between the system's (plant) variables, including constraints such as input limits and state/output ranges. An optimization problem is formulated based on these constraints and an objective (cost) function. At each sampling interval, the optimization problem is solved with the current state serving as the initial condition, and with respect to reference variables for tracking a desired output trajectory. The optimal control problem is formulated over a time interval starting from the current time and extending for a defined duration into the future. The outcome of this optimization is a sequence of future control actions. This iterative process is repeated at each subsequent time step. A general MPC structure is shown in Fig. 2.2.

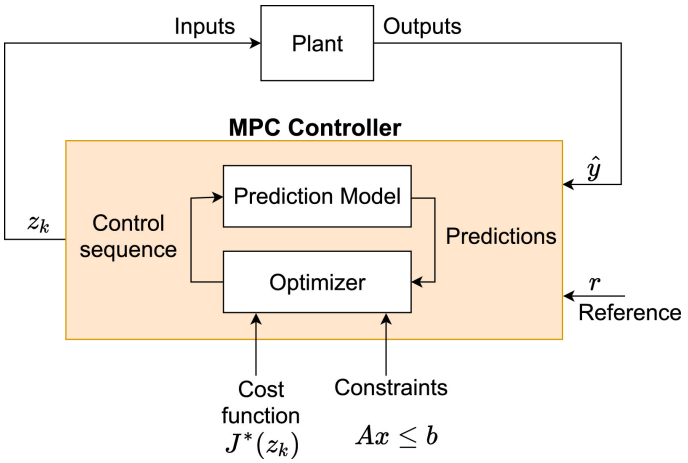


Figure 2.2: A general MPC structure, source from [5]

Model representation

MPC is usually based on a linear discrete-time prediction model that describes the system's dynamics, typically expressed as:

$$x(t+1) = Ax(t) + Bu(t), \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector at time t, $u(t) \in \mathbb{R}^m$ is the control input vector, and matrices A and B define the system dynamics. Although simple and efficient, linear MPCs (LMPCs) can only approximate the behaviour of nonlinear systems accurately around a specific operating point. Also, LMPCs are not robust and perform poorly if there are disturbances or dynamic changes in the system. For example, in autonomous racing, LMPC might perform well when the car is driving on a straight path. However, its performance can degrade significantly when the car takes sharp turns or encounters track conditions such as changes in elevation or friction changes due to weather conditions. Therefore, other variants of MPCs are used instead:

- **Nonlinear MPC (NMPC)** – directly incorporates nonlinear dynamics by using a nonlinear model of the system. This offers better accuracy and performance compared to LMPC [6]. However, NMPC is computationally more demanding than LMPC due to the complexity of solving nonlinear optimization problems.

- **Hybrid MPC** – manages systems that have both continuous dynamics (like speeds and temperatures) and discrete events (like switches turning on and off) [7]. It han-

dles situations where the system can switch between different modes of operation. For example, in an automotive system, hybrid MPC can manage both the continuous speed of the car and the discrete gear shifts. The drawback of hybrid MPC is its complex implementation.

- **Adaptive MPC** – adjusts parameters in real-time to handle changes in the system. Unlike LMPC, adaptive MPC updates the state-space matrices over time. This allows flexibility under varying conditions [8], but it may be slower because of the continuous need to update the model. Consequently, if these changes happen too fast, adaptive MPC may not keep up.

Formulation of the Optimization Problem

The optimization problem at each sampling interval tries to find control inputs that minimize a cost function over a prediction horizon N. The cost function is expressed as:

$$\begin{aligned} \min_{U} J &= x_N^T P x_N + \sum_{k=0}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k \right) \\ \text{s.t. } x_{k+1} &= A x_k + B u_k, \quad k = 0, \dots, N-1, \\ x_0 &= x(t), \\ u_{\min} &\leq u_k \leq u_{\max}, \quad k = 0, \dots, N-1, \\ y_{\min} &\leq C x_k \leq y_{\max}, \quad k = 1, \dots, N. \end{aligned} \tag{2.2}$$

where x_k and u_k are the state and control input at step k of the prediction horizon. The terminal matrix P is used to penalize the state at the final time step N of the prediction horizon, and it is often chosen as the solution of the Riccati equation for infinite N [9]. The cost function also includes two key components: Q and R.

1. **Weight Matrix** Q – penalizes deviations between the system's state and the desired state. For autonomous vehicles, Q can prioritize maintaining a smooth and safe trajectory, minimizing errors in position, lateral and longitudinal speeds, and other relevant variables.

2. **Weight Matrix** R – penalizes control inputs. In the case of autonomous vehicles, R ensures that the control actions (such as wheel torques) are neither too aggres-

sive nor too conservative. This helps in reducing wear and tear on the vehicle or conserving energy.

If Q is too high compared to R, the vehicle will prioritize following the exact path but might make aggressive control actions, leading to uncomfortable or unsafe driving. On the other hand, if R is too high compared to Q, the vehicle will prioritize smooth control actions but might deviate more from the reference path. MPC can also include penalties on the derivatives of control inputs to prevent rapid changes that could damage the vehicle's components.

Receding Horizon Control

MPC operates on a receding horizon principle. At each time step t, the current state $x(t)$ is used to solve the optimization problem. The first control action $u(t) = u_0^*$ is applied to the system, and the horizon is shifted forward by one step, repeating this process at each time step. The strategy is illustrated in Fig. 2.3.

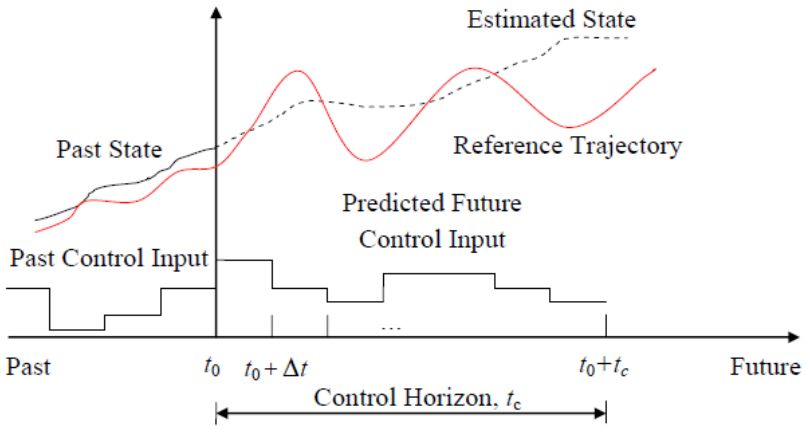


Figure 2.3: Illustration of MPC receding horizon strategy, source from [10]

A longer prediction horizon improves the accuracy of future behaviour predictions but also increases computational complexity, making real-time response more challenging. This is critical for MPCs that need to operate online and adapt dynamically to changing conditions.

2.2 MPC in Autonomous Racing

For this master's thesis, an existing MPC control system developed by [11] in Simulink was utilized for testing purposes. The MPC Toolbox in MATLAB was used to construct the controller [12]. The graphical user interface (GUI) allows designing, simulating, and tuning of MPC controllers, which can then be exported to MATLAB as MPC objects. These objects can afterwards be easily integrated into Simulink models using the Simulink library. The whole test setup in Simulink is shown in Fig. 2.4 and it consists of the following blocks:

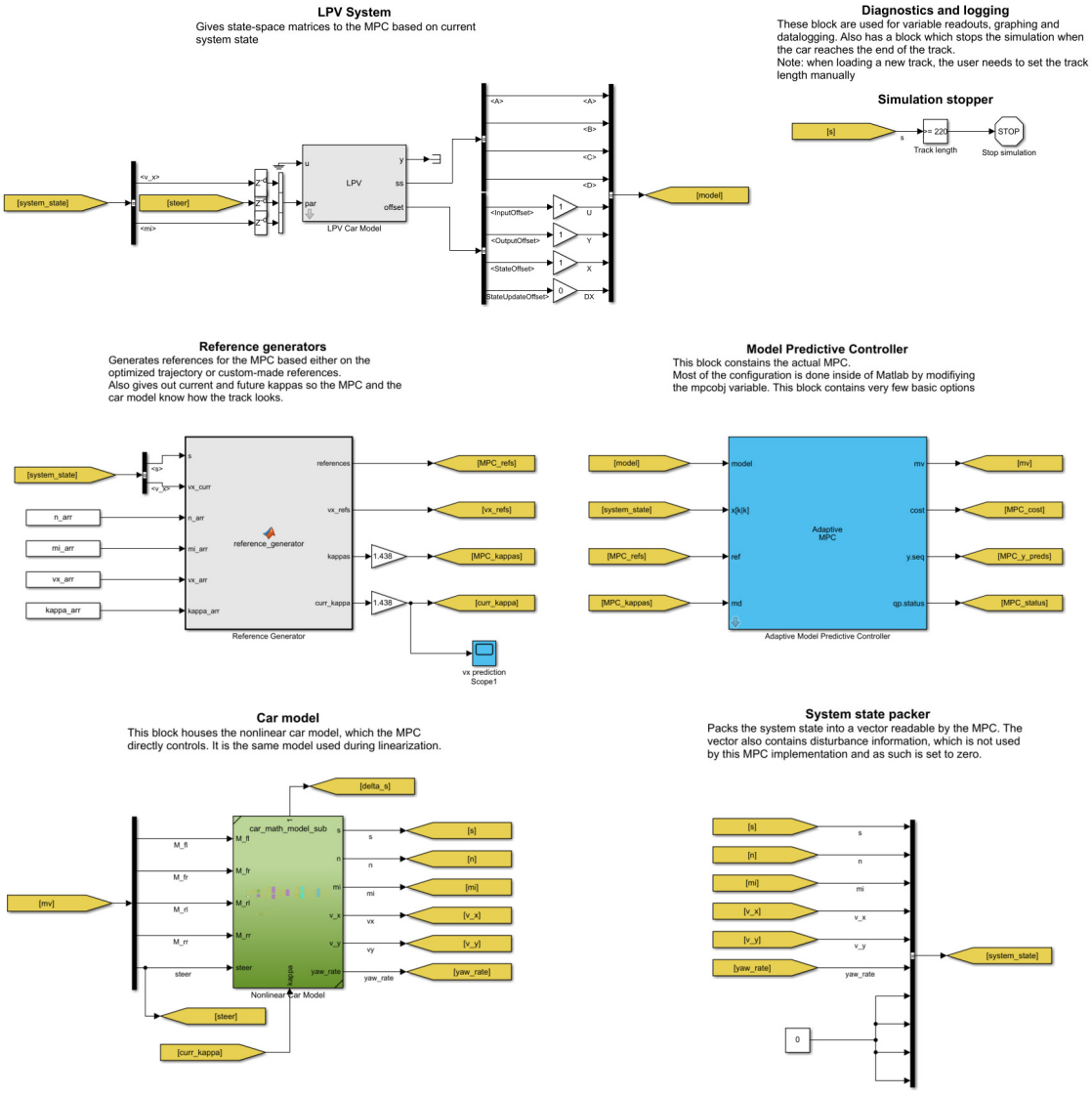


Figure 2.4: Experimental MPC setup in Simulink, source from [11]

Controller

An adaptive MPC is implemented for use as a controller. The controller utilizes a Linear Time-Invariant (LTI) discrete model obtained through linearization, along with the current system state vector, reference values for the objective function, and measured disturbances (track curvature). Its output is a vector of control variables. At the beginning of each control interval, a bus signal initiates an update of the plant model prediction and the nominal operating point. The adaptive mechanism enables the controller to dynamically adjust to changes in the nonlinear system's behaviour over time [13]. The adaptive MPC block uses an MPC object created beforehand. The state models of the MPC object are chosen as:

- s [m]: Distance traveled along the track from the starting point.
- n [m]: Lateral displacement from the track centerline at position s.
- μ [rad]: Vehicle's angular deviation from the track centerline at position s.
- v_x [m/s]: Longitudinal speed of the vehicle in its local coordinate system.
- v_y [m/s]: Lateral speed of the vehicle in its local coordinate system.
- ψ' [rad/s]: Yaw rate, indicating the rate of change of the vehicle's orientation around the vertical axis.

The input models are:

- M [Nm]: Torque applied to a specific wheel.
- δ [rad]: The steering angle of the front wheels.

Reference generator

The MPC reference trajectory is generated from a pre-existing offline time-optimized path. The current position along the curved path system is determined based on the current s and v_x. Then, the next N references are generated for the prediction horizon from this current position.

LPV System

To obtain a linear model represented by matrices A, B, C, D as the input for the MPC controller, the nonlinear plant model must first be linearized. This was achieved using the Linear Parameter Varying System (LPV) block in MATLAB [14]. The LPV block takes the current system state as an operational input point. An LPV model consists of an interpolated array of linear state-space models. A set of points in the operational space are chosen, forming a regular grid. For each point, an LTI system is constructed, representing the local dynamics at that exact condition.

When the system operates between these predefined grid points, interpolation techniques are applied instead of creating a new model from scratch. This involves estimating the system's behaviour by interpolating between LTI systems at neighboring grid points. Before using the LPV block, an LPV object had to be created, and the ranges of operational points for the states had to be defined. Linearization was performed prior to running the simulation, so that in real-time only interpolation and retrieval of the LTI system matrices are performed.

To further illustrate the versatility of LPV modelling, [15] demonstrates its application in diverse systems such as diesel engines, wind turbine control, and aircraft modelling and control. This showcases LPV's capability to handle complex nonlinear systems with varying operating conditions.

Nonlinear Plant Model

Control actions generated by the MPC controller are directly applied to the plant model, which simulates a four-wheeled car. The model incorporates factors like aerodynamic forces, tire dynamics, and rolling resistance to provide a realistic representation of the vehicle's behaviour. The output of the plant model is the vehicle state at the next time step. The simulation runs in discrete steps of $\Delta t = 0.025$ seconds and ends when the vehicle reaches the end of the path, that is, when the total path length is surpasses 220 meters ($s \geq 220$ m).

3 Stochastic Optimization Methods

Traditional optimization methods for decision-making typically assume perfect information, where relevant system parameters are accurately known and stochastic variables follow well-defined probability distributions, such as Gaussian distributions with known means and variances. However, such precision is rarely achievable in real-world scenarios, where decision-makers often rely on noisy historical data. Stochastic optimization methods address this uncertainty by integrating random variables directly into the optimization process. In stochastic optimization, randomness is an inherent part of the problem formulation, either within the objective function or constraints.

There are numerous advantages and applications of stochastic optimization, especially in situations where experimental measurements are subject to random errors. In scenarios where noise affects the data, it is practical to employ algorithms that use statistical methods for estimating the true values or objectives. This way, statistically optimal decisions are created for the next stages in the optimization process. Additionally, incorporating randomness into the search process can speed up the progress [16]. Furthermore, the model can become more robust to inadvertently formulated errors and reduce the risk of getting trapped in local optima, instead potentially guiding the search towards discovering the global optimum. The stochastic problem can be formulated as:

$$\min_{x \in X} \ \mathrm{Exp}_p\{f(x, \xi)\}. \tag{3.1}$$

The goal is to minimize the expected value of the objective function $f(x, \xi)$ with respect to the decision variable x, where x belongs to the set X, and ξ represents uncertain parameters. The minimization of the expectation is necessary because the exact probabilistic distribution of ξ is often unknown to accurately estimate in real-world scenarios. By

minimizing the expectation, the optimization process aims to find a solution that performs well on average across potential scenarios.

Stochastic vs Deterministic Optimization

Apart from a stochastic approach, optimization can also be done deterministically, each approach offering distinct advantages suited to different types of problems.

Deterministic methods excel in situations where the goal is to find the globally optimal solution with theoretical guarantees. They are effective when the problem is well-defined, and the objective function is known or easily accessible. Examples include Integer Programming (IP), Non-convex Nonlinear Programming (NNLP), and Mixed-Integer Nonlinear Programming (MINLP). However, they may struggle when tackling black-box problems where the function behaviour is not well-understood or lacks smoothness, making it challenging to converge to an optimal solution. They can face challenges with complex large-scale problems, as the sheer number of variables and constraints can lead to a combinatorial explosion.

In contrast, stochastic methods do not provide guarantees for finding the global optimum but offer probabilistic completeness. This means the probability of finding the globally optimal solution increases with computational time and approaches 100% in infinite time. This makes stochastic optimization more suitable for situations where reaching a satisfactory solution within a feasible time frame is more important than finding the best solution. Stochastic methods are often more computationally efficient and adaptable, making better use of CPU power. That is because they can easily be parallelized and often utilize sample-based algorithms to explore high-dimensional or large-scale search spaces.

In summary, deterministic optimization is preferred when the primary goal is to obtain the globally optimal solution, while stochastic optimization is useful for scenarios where simply finding a good enough solution within a given time frame is sufficient. Table 3.1 shows the difference between these two approaches, highlighting their characteristics and uses.

Table 3.1: Stochastic and deterministic optimization comparison, source from [17]

	Deterministic Optimization	**Stochastic Optimization**
Globally Optimal Result	Guaranteed	Guaranteed only with infinite execution time
Execution Time	May be long for big scale problems	Execution time is controllable depending on user's necessities
Problem Models	LP, NP, NMLP, NLP, MINLP	Any
Algorithm Examples	Primal-Dual Decomposition, Reverse Convex, Cutting Plane	Genetic algorithms, stochastic gradient descent, particle swarm optimization

3.1 Cross-Entropy Method

The cross-entropy (CE) method is a Monte Carlo approach used for importance sampling and optimization. It relies on two iterative phases: sampling from a probability distribution and minimizing the cross-entropy between the true distribution f and a target distribution p parameterized by θ. The CE method aims to minimize the expression:

$$-\int_x f(x) \log p(x \mid \theta)\, dx. \tag{3.2}$$

The goal is to find events where the objective function S exceeds a threshold γ. The probability of this event can be expressed as the expectation:

$$\ell = \mathbb{E}_\theta[\mathbb{1}_{S(x)\geq\gamma}]. \tag{3.3}$$

For rare events, this estimation can be difficult. Importance sampling addresses this challenge by using a different sampling distribution to sample more efficiently from the regions where the rare event occurs, improving the estimation process. The optimal importance sampling density is:

$$f^*(x) = \frac{\mathbb{1}_{S(x)\geq\gamma} p(x \mid \theta)}{\ell} \tag{3.4}$$

The indicator function $\mathbb{1}_{S(x)\geq\gamma}$ ensures only the regions where $S(x) \geq \gamma$ are considered, and ℓ normalizes this density. Substituting $f^*(x)$ in Eq. 3.2 leads to the following opti-

mization problem:

$$\begin{aligned}\theta_p^* &= \arg\min_{\theta_p}\left(-\int_{x\in X}\mathbb{1}_{S(x)\geq\gamma}p(x\mid\theta)\log p(x\mid\theta_p)\,dx\right)\\ &= \arg\min_{\theta_p}\left(-\mathbb{E}_\theta\left[\mathbb{1}_{S(x)\geq\gamma}\log p(x\mid\theta_p)\right]\right).\end{aligned} \tag{3.5}$$

In [18], different variants of CEM optimization were discussed. This thesis implements the following approaches in MATLAB: CEM with multivariate Gaussian and Gaussian mixtures as target distributions, and CEM with surrogate models for prediction of rare events.

3.1.1 Multivariate Gaussian Distribution Model

If the distribution $p(\mathbf{x}\mid\theta_p)$ is normal, then calculating the optimal parameters for θ_p involves finding the mean ($\boldsymbol{\mu}$) and the covariance matrix ($\boldsymbol{\Sigma}$). To do this, the Maximum Likelihood Estimation (MLE) method is used. For a multivariate Gaussian distribution, the probability density function is given by:

$$p(\mathbf{x}\mid\theta_p)=\frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}. \tag{3.6}$$

Minimizing the likelihood function (MLE) with respect to these parameters is equivalent to minimizing the negative log-likelihood function. The likelihood is expressed as:

$$L(\boldsymbol{\mu},\boldsymbol{\Sigma})=\prod_{i=1}^{m_{\text{elite}}}p(\mathbf{x}_i\mid\theta_p), \tag{3.7}$$

and the negative log-likelihood evaluates to:

$$\log L(\boldsymbol{\mu},\boldsymbol{\Sigma})=-\frac{m_{\text{elite}}\cdot n}{2}\log(2\pi)-\frac{m_{\text{elite}}}{2}\log|\boldsymbol{\Sigma}|-\frac{1}{2}\sum_{i=1}^{m_{\text{elite}}}(\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}). \tag{3.8}$$

The partial derivatives of this function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are calculated and set to zero to obtain the optimal parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\frac{\partial}{\partial\boldsymbol{\mu}}\log L(\boldsymbol{\mu},\boldsymbol{\Sigma})=0\quad\rightarrow\quad\hat{\boldsymbol{\mu}}_{\text{ML}}=\frac{1}{m_{\text{elite}}}\sum_{i=1}^{m_{\text{elite}}}\mathbf{x}_i \tag{3.9}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \log L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0 \quad \rightarrow \quad \hat{\boldsymbol{\Sigma}}_{\mathrm{ML}} = \frac{1}{m_{\mathrm{elite}}} \sum_{i=1}^{m_{\mathrm{elite}}} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{\mathrm{ML}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{\mathrm{ML}})^T \tag{3.10}$$

Algorithm 1 outlines the pseudocode of this method. The class is initialized with values for the means ($\boldsymbol{\mu}_{\mathrm{init}}$) and covariances ($\boldsymbol{\Sigma}_{\mathrm{init}}$). These values are arbitrarily chosen but should provide a good initial guess about the solution, so the final solution does not end up in a local optimum. The algorithm runs for $iter_{max}$ iterations.

In each iteration n_s instances (denoted as $\mathbf{X}$) are sampled from a normal distribution defined with the current $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In MATLAB, `mvnrnd` function is used for this purpose. A parallel for loop is used to process each sample in $\mathbf{X}$. For each sample, a simulation of the system (`sys`) is applied to obtain the data, and an objective function is evaluated on the resulting data. The objective function could, for example, be the sum of the parameters in the data. The results are stored in $\mathbf{Y}$. After evaluating all samples, $\mathbf{Y}$ is sorted based on the chosen order (either ascending or descending). The top n_e samples from $\mathbf{X}$, based on the sorted $\mathbf{Y}$, are selected and stored in e. At the end of each iteration, new values for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are calculated from e using Maximum Likelihood (ML) estimates from Eq. 3.9 and Eq. 3.10.

Algorithm 1 Cross Entropy Method with Normal Distribution

```
1:  Class CEMnormal
2:    Constructor: (μ_init, Σ_init, iter_max, n_s, n_e, f, order, sys)
3:      μ ← μ_init
4:      Σ ← Σ_init
5:
6:    Function Run()
7:      for i = 1 to iter_max do
8:        X ← SampleFromNormalDistribution(μ, Σ, n_s)
9:        Y ← init(n_s)
10:       parfor n = 1 to n_s do
11:         data ← SimulateData(X_n, sys)
12:         Y_n ← EvaluateObjective(f, data)
13:       end parfor
14:       Y ← Sort(order)
15:       e ← SelectTopSamples(X, Y, n_e)
16:       μ, Σ ← FitNormalDistribution(e)
17:     end for
18:     Return μ, Σ
19:
20:   End Function
21: End Class
```

3.1.2 Mixture of Gaussians Model

Mixture of Gaussians

A standard Gaussian distribution is unimodal and may struggle with capturing data that exhibits multimodal behaviour. In contrast, a Gaussian Mixture Model (GMM) is a weighted combination of component distributions, where the probability density function $p(\mathbf{x})$ is a linear combination of K Gaussian probability density functions. Each example is assigned to a group with probability $h_k^{(i)}$, ranging between 0 and 1. The probability density function of a GMM can be expressed as:

$$\begin{aligned} p(\mathbf{x}) &= \sum_{k=1}^{K} P(\mathbf{x}, y = k) = \sum_{k=1}^{K} P(y = k) p(\mathbf{x} \mid y = k) = \sum_{k=1}^{K} \pi_k p(\mathbf{x} \mid \theta_k) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \end{aligned} \tag{3.11}$$

where π_k is the prior probability of a group, with the constraint $\sum_k \pi_k = 1$. The log-likelihood of probability densities can be represented as:

$$\begin{aligned} \ln L(\boldsymbol{\theta} \mid D) &= \ln \prod_{i=1}^{N} p\left(\mathbf{x}^{(i)}\right) = \ln \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k p\left(\mathbf{x}^{(i)} \mid \theta_k\right) \\ &= \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k p\left(\mathbf{x}^{(i)} \mid \theta_k\right). \end{aligned} \tag{3.12}$$

The parameters of the model that need to be estimated are thus $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$.

Expectation-Maximization algorithm

Maximizing the log-likelihood doesn't have a closed-form solution since its derivatives cannot be factored with respect to its components, which is why the Expectation-Maximization (EM) algorithm is used. The concept of this algorithm is to introduce hidden variables which enable calculating the expected log-likelihood with fixed values for parameters π_k and $\boldsymbol{\theta}_k$. Then, these parameters are updated to maximize this expectation, with resulting

expressions given in Eq. 3.13. This process repeats iteratively.

$$\begin{aligned}
\mu_k &= \frac{\sum_i h_k^{(i)} \mathbf{x}^{(i)}}{\sum_i h_k^{(i)}}, \\
\Sigma_k &= \frac{\sum_i h_k^{(i)} (\mathbf{x}^{(i)} - \mu_k)(\mathbf{x}^{(i)} - \mu_k)^T}{\sum_i h_k^{(i)}}, \\
\pi_k &= \frac{1}{N} \sum_{i=1}^{N} h_k^{(i)}.
\end{aligned} \tag{3.13}$$

The pseudocode of the method is shown in Algorithm 2. In the first step of the algorithm (E-step), the probability a sample $\mathbf{x}^{(i)} \in D$ belongs to a group $k \in K$ is calculated. h_k can be derived using Bayes' Rule:

$$\begin{aligned}
h_k^{(i)} &= P\left(y = k \mid \mathbf{x}^{(i)}\right) \\
&= \frac{P\left(y = k\right) p\left(\mathbf{x}^{(i)} \mid y = k\right)}{p\left(\mathbf{x}^{(i)}\right)} \\
&= \frac{P\left(y = k\right) p\left(\mathbf{x}^{(i)} \mid y = k\right)}{\sum_j P\left(y = j\right) p\left(\mathbf{x}^{(i)} \mid y = j\right)} \\
&= \frac{\pi_k p\left(\mathbf{x}^{(i)} \mid \theta_k\right)}{\sum_j \pi_j p\left(\mathbf{x}^{(i)} \mid \theta_j\right)}.
\end{aligned} \tag{3.14}$$

In the second step (M-step), new parameters for the groups are computed based on the current assignment of examples to groups, following expressions 3.13. The algorithm iterates until either the parameters stabilize or the log-likelihood converges.

Algorithm 2 Gaussian Mixture Model (GMM) Algorithm, source [19]

1: **Initialization:** $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$
2: **Repeat until convergence of θ or $L(\theta|D)$:**
3: **E-step:**
4: Calculate $h_k^{(i)}$ by Eq. (3.14)
5: **M-step:**
6: Calculate μ_k, Σ_k, π_k by Eq. (3.13)
7: Calculate $L(\theta|D)$

Implementation

Using GMMs in optimization tasks offers several advantages over single Gaussian distributions:

- **Enhancing exploration-exploitation tradeoff**: During optimization, some components may explore new regions of the search space while others exploit promising areas. This approach helps GMMs avoid converging to local optima, as illustrated in Fig. 3.1.
- **Flexibility**: GMMs can be useful for identifying patterns or groups within the data by breaking it down into separate modes. This allows a better understanding of the data structure and behaviour.
- **Outlier robustness**: Outliers can affect the parameters estimated by single Gaussians, which leads to biased results. Due to GMMs' capacity to distribute the data among multiple components, the impact of outliers on the overall model is reduced.

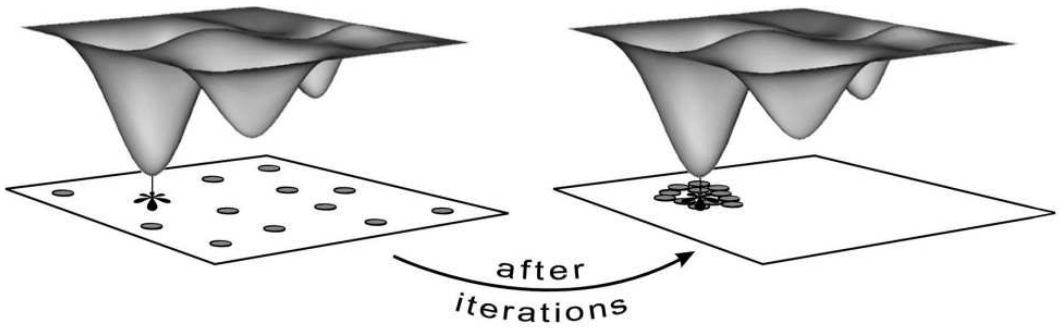


Figure 3.1: GMM exploration and convergence with multiple clusters, source from [20]

The method was implemented similarly to CEM described in Subsection 3.1.1. The class is initialized with the same properties as there, but now the means ($\boldsymbol{\mu}$) and variances ($\boldsymbol{\Sigma}$) are matrices of size $n_{\text{cluster}} \times d$. Each row of these matrices represents a d-dimensional cluster. Additionally, the prior probabilities of the components (π) are initialized. The difference from Algorithm 1 is in lines 8 and 16. Now, instances are sampled from a Gaussian mixture distribution, defined by the current parameters $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and π. In line

6 of Algorithm 3, the GMM is created as an object in MATLAB using `gmdistribution`, and sampling is performed using the `random` function. In line 7, new parameter values are calculated from the elite weights. After that, using MATLAB's built-in `fitgmdist` function, the GMM parameters are updated based on the EM algorithm.

Algorithm 3 Cross Entropy Method with Gaussian Mixture Model (GMM)

```
1: Class CEMgmm
2:     Constructor: (..., π)
3:
4:     Function Run()
5:         ...
6:         X ← SampleFromGMM(μ, Σ, n_s, π)
7:         μ, Σ, π ← FitGMM(e)
8:         ...
```

3.1.3 Surrogate Model

In optimization, surrogate models are used to approximate the true objective function, enabling less expensive evaluations. This is useful for estimating rare events in systems that require a lot of computational resources or take a long time to simulate. Fitting a surrogate model involves training the model on data to accurately represent the true objective function. This process includes selecting the appropriate model type, such as linear or logistic regression. In this method, Gaussian processes are used.

Gaussian Processes

A Gaussian process (GP) is a stochastic process where any finite number of random variables follows a multivariate normal distribution. Unlike models composed of a fixed number of functions, a GP defines a distribution over all possible functions that fit the random data (Fig. 3.2). A GP is defined by a mean vector and a covariance matrix, which are typically constructed using kernels. Kernels are preferred over classical covariance functions because they can capture a wider range of correlations, including non-linear relationships. This allows GPs to:

- Provide uncertainty estimates in predictions,
- Model non-linearity,
- Incorporate prior knowledge,

- Be scalable to higher dimensions.

Examples of kernels include the Radial Basis Function (RBF) kernel, the periodic kernel, and the linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$, which corresponds to classical covariance functions. Another advantage of Gaussian processes over traditional regression models is that they are non-parametric. This means they can adapt to the data without assuming a fixed model structure.

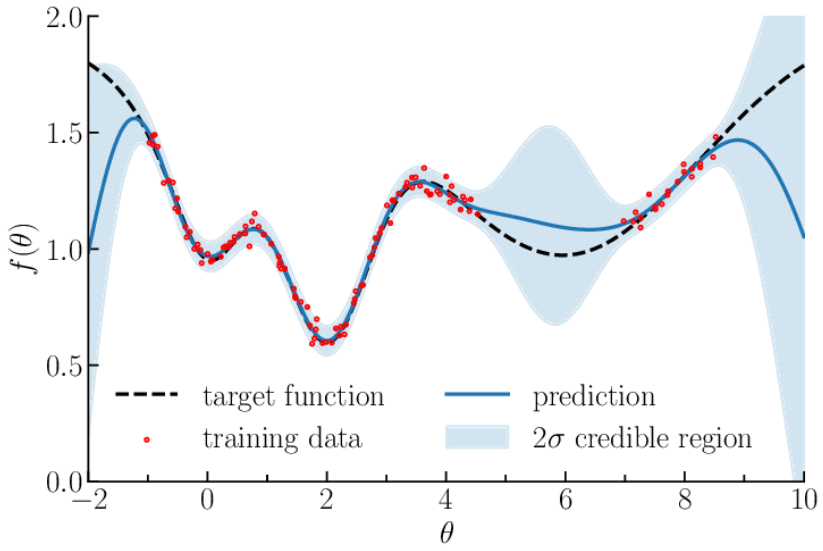


Figure 3.2: Gaussian processes for regression, source from [21]

Consider a training set with observations $\{(x_i, y_i)\}_{i=1}^n$, where $y_i = f(x_i) + \epsilon_i$. Here ϵ_i represents an independent Gaussian noise with ($\epsilon_i \sim \mathcal{N}(0, \sigma^2)$). The input and output vectors are denoted as $\mathbf{X} = [x_1, x_2, \dots, x_n]^T$ and $\mathbf{Y} = [y_1, y_2, \dots, y_n]^T$.
GP can be used to predict a new test set $(\mathbf{X}^*, \mathbf{Y}^*)$. The joint distribution of the training and test sets is also Gaussian. Assuming the mean vectors have a zero value, the joint distribution derives to:

$$\begin{aligned}
\begin{bmatrix} Y \\ Y^* \end{bmatrix} &= \begin{bmatrix} f(X) \\ f(X^*) \end{bmatrix} + \begin{bmatrix} \epsilon \\ \epsilon^* \end{bmatrix} \\
&\sim \mathcal{N}\left(0, \begin{bmatrix} k(X,X) & k(X^*,X) \\ k(X,X^*) & k(X^*,X^*) \end{bmatrix}\right) + \mathcal{N}\left(0, \begin{bmatrix} \sigma^2 I & 0 \\ 0 & \sigma^2 I \end{bmatrix}\right) \\
&= \mathcal{N}\left(0, \begin{bmatrix} k(X,X) + \sigma^2 I & k(X^*,X) \\ k(X,X^*) & k(X^*,X^*) + \sigma^2 I \end{bmatrix}\right).
\end{aligned} \tag{3.15}$$

In estimation theory, the lemma concerning the conditional distribution of a multivariate normal distribution is often utilized when predicting a set of variables given another set [22]. This theorem states that if the random variables y and x follow a joint Gaussian distribution:

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}\right). \tag{3.16}$$

Then the conditional probability of y given x is:

$$p(y|x) = \mathcal{N}\left(\bar{y} + CA^{-1}(x - \bar{x}), B - C^T A^{-1} C\right). \tag{3.17}$$

Now, using this theorem and defining the conditional distribution:

$$P(Y^* \mid Y) \sim \mathcal{N}(\mu, \Sigma). \tag{3.18}$$

The predictive mean and covariance evaluate to:

$$\begin{aligned} \mu^* &= k(X^*, X)\left(k(X, X) + \sigma^2 I\right)^{-1} Y, \\ \Sigma^* &= k(X^*, X^*) + \sigma^2 I - k(X, X^*)\left(k(X, X) + \sigma^2 I\right)^{-1} k(X^*, X). \end{aligned} \tag{3.19}$$

Implementation

The class for CEM with Surrogate Models was implemented similarly to Algorithm 1, but with a difference in the parameters' update. In this implementation (line 7 in Algorithm 4), not only true-elite samples (e) from simulations are used for updating the parameters, but an elite set is modelled. This elite set (E) includes samples e_m and e_{sub}, in addition to the true-elites mentioned earlier.

Algorithm 4 Cross Entropy Method with Surrogate Model

```
1: Class CEMsurrogate
2:     Constructor: (..., iter_CEM)
3:
4:     Function Run()
5:         ...
6:         e ← SelectTopSamples(X, Y, n_e)
7:         E ← ModelEliteSet(X, Y, e, μ, Σ)
8:         μ, Σ ← FitNormalDistrribution(E)
9:         ...
```

The model-elites e_m are estimated in Algorithm 5. Initially, the surrogate model $\hat{S}$ is constructed from the samples $\mathbf{X}$ and true objective function values $\mathbf{Y}$, using Gaussian Processes. In MATLAB, the function `fitrgp` is used for this purpose. Afterwards, $10 \cdot n_s$ new instances are sampled ($\hat{\mathbf{X}}$) and evaluated using the surrogate model. The predictions $\hat{\mathbf{Y}}$ are then used to filter the top $10 \cdot n_e$ samples. These are used as the model-elites.

Algorithm 5 Model EliteSet Function

Function ModelEliteSet($\mathbf{X}$, $\mathbf{Y}$, e, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)
  $\hat{S} \leftarrow$ FitGP($\mathbf{X}$, $\mathbf{Y}$)
  $\hat{\mathbf{X}} \leftarrow$ SampleFromNormalDistribution($\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $10 \cdot n_s$)
  $\hat{\mathbf{Y}} \leftarrow$ Predict($\hat{S}$, $\hat{\mathbf{X}}$)
  $\hat{\mathbf{Y}} \leftarrow$ Sort($order$)
  $e_m \leftarrow$ SelectTopSamples($\hat{\mathbf{X}}$, $\hat{\mathbf{Y}}$, $10 \cdot n_e$)
  $e_{sub} \leftarrow$ ModelSubEliteSet($\hat{S}$, e, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $iter_{CEM}$)
  $E \leftarrow$ Concatenate(e, e_m, e_{sub})
  return E
End Function

The sub-elites e_{sub} are generated using the true-elites, with the goal to exploit their information to focus on areas of the data that show the most potential. Each true elite serves as a mean vector for sampling new instances ($\hat{\mathbf{x}}_{sub}$), with the inherited covariance from Algorithm 4. These new samples are then assessed using the surrogate model in a standard CEM method, where the parameters are updated for $iter_{CEM}$ iterations. The resulting top samples are used as e_{sub}, as shown in Algorithm 6.

Algorithm 6 Model SubEliteSet Function

Function ModelSubEliteSet($\hat{S}$, e, $\boldsymbol{\Sigma}$, $iter_{CEM}$)
  $e_{sub} \leftarrow \emptyset$
  for $n = 1$ to n_e **do**
    $\boldsymbol{\mu}_n \leftarrow \mu(e_n)$
    $\hat{\mathbf{x}}_{\text{sub}} \leftarrow$ SampleFromNormalDistribution($\boldsymbol{\mu}_n$, $\boldsymbol{\Sigma}$, n_s)
    $e_{nsub} \leftarrow$ CEMNormal($\hat{S}$, $\hat{x}_{\text{sub}}$, $iter_{CEM}$)
    $e_{sub} \leftarrow e_{sub} \cup \{e_{nsub}\}$
  end for
  return e_{sub}
End Function

3.2 Bayesian Optimization

Bayesian Optimization is another method used for optimizing complex, expensive, time-consuming functions. It builds a probability model of the objective function and uses it to select hyperparameters to evaluate the actual function. The key parts of Bayesian optimization are: the true objective function, a surrogate model and an acquisition function. A surrogate model approximates the objective and estimates its uncertainty using a Bayesian machine learning technique (like Gaussian Processes), while acquisition functions guide the search for the next point to evaluate. A tutorial on Bayesian optimization with constraints was given in [23].

Acquisition functions

Acquisition functions decide points where to evaluate next in the search space. By utilizing predictions from the surrogate model and considering the uncertainty from those predictions, acquisition functions identify points that are likely to have high objective function values or areas where there is considerable uncertainty, suggesting potential for improvement. This way they can balance exploration and exploitation and intelligently search the space. Additionally, acquisition functions are cheaper to evaluate than the original objective function, making them more computationally feasible to use in optimization. Common acquisition functions include:

- Expected Improvement (EI) – chooses points that are expected to improve over the current best-known value. EI tends to explore more diverse regions early on but becomes more exploitative as it gets closer to the optimal solution.

$$\mathrm{EI} = \left(y_{\mathrm{pred}} - \mathrm{best\_y}\right) \cdot \mathrm{cdf}(z) + y_{\mathrm{std}} \cdot \mathrm{pdf}(z) \tag{3.20}$$

  where

$$z = \frac{y_{\mathrm{pred}} - \mathrm{best\_y}}{y_{\mathrm{std}}}.$$

- Probability of Improvement (PI) – picks points based on the chance that they will surpass the current best-known value, but focuses solely on the probability of improvement. PI tends to prefer points where there is a high probability of improve-

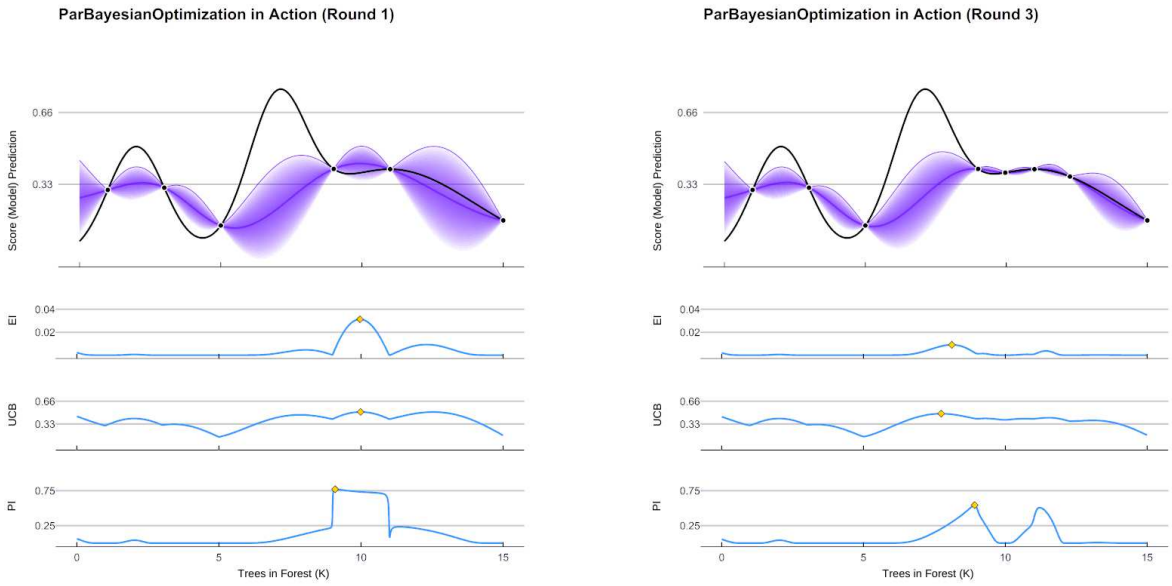
ment regardless of whether this improvement is small or large.

$$\mathrm{PI} = \mathrm{cdf}(z) \tag{3.21}$$

- Upper Confidence Bound (UCB) – selects points that balance between promising high-value regions and unexplored or uncertain areas.

$$\mathrm{UCB} = y_{\mathrm{pred}} + \beta \cdot y_{\mathrm{std}} \tag{3.22}$$

Examples of acquisition functions over rounds are shown in Fig. 3.3. The first graph depicts the surrogate model, followed vertically by EI in the second, UCB in the third, and PI in the last.



(a) Iteration 1 **(b)** Iteration 3

Figure 3.3: Acquisition functions through iterations, source from [24]

Once the acquisition function selects the next hyperparameter, its performance is measured using the objective function. This provides new data to update the surrogate model and the cycle repeats until either the maximum time or the maximum number of iterations is reached.

Implementation

The algorithm for Bayesian optimization, shown in Algorithm 7, begins by initializing means and covariances. Instances are then sampled from a normal distribution. Additionally, n_{UCB} instances ($\mathbf{X}_{\text{UCB}}$) are sampled to define the search space for the acquisition function. The data is simulated and evaluated as before, in a parallel for loop. Following this initialization, the algorithm enters a for loop where, in each iteration, a surrogate model (here Gaussian Processes), is constructed or updated based on the samples $\mathbf{X}$ and true objective function values $\mathbf{Y}$. Next, the acquisition function selects a new point ($\mathbf{X}_{\text{new}}$) to evaluate. This new point is then evaluated in the actual system simulation to obtain its objective function value $\mathbf{Y}_{\text{new}}$. The set ($\mathbf{X}_{\text{new}}, \mathbf{Y}_{\text{new}}$) is added to the vectors $\mathbf{X}$ and $\mathbf{Y}$, so that the surrogate model can be updated by incorporating the latest information. The cycle is repeated for $iter_{max}$ iterations, continually refining the surrogate model and identifying optimal points in the search space.

Algorithm 7 Bayesian Optimization

Class BayesianOptimization
  Constructor: BayesianOptimization($\boldsymbol{\mu}_{\text{init}}$, $\boldsymbol{\Sigma}_{\text{init}}$, $iter_{max}$, n_s, n_{UCB}, f, sys, β)
    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}_{\text{init}}$
    $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma}_{\text{init}}$

  Function Run()
    $\mathbf{X} \leftarrow$ SampleFromNormalDistribution($\boldsymbol{\mu}, \boldsymbol{\Sigma}, n_s$)
    $\mathbf{X}_{\text{UCB}} \leftarrow$ SampleFromNormalDistribution($\boldsymbol{\mu}, \boldsymbol{\Sigma}, n_{\text{UCB}}$)
    $\mathbf{Y} \leftarrow$ init(n_s)
    for $n = 1$ to n_s **do**
      $data \leftarrow$ SimulateData(X_n, sys)
      $\mathbf{Y}_n \leftarrow$ EvaluateObjective($f, data$)
    end for

    for $i = 1$ to $iter_{max}$ **do**
      $\hat{S} \leftarrow$ FitGP(X, Y)
      $\mathbf{X}_{\text{new}} \leftarrow$ UCB($\mathbf{X}_{\text{UCB}}, \hat{S}, \beta$)
      $data \leftarrow$ SimulateData($\mathbf{X}_{\text{new}}$, sys)
      $\mathbf{Y}_{\text{new}} \leftarrow$ EvaluateObjective($f, \mathbf{X}_{\text{new}}$)
      $\mathbf{X} \leftarrow$ Add($\mathbf{X}_{\text{new}}$)
      $\mathbf{Y} \leftarrow$ Add($\mathbf{Y}_{\text{new}}$)
    end for

    Return $\mu, \boldsymbol{\Sigma}$
  End Function
End Class

Algorithm 8 Upper Confidence Bound (UCB)

1: **Function** UCB($\mathbf{X}_{UCB}$, $\hat{S}$, β)
2:
3: $\mathbf{Y}_{pred}, \mathbf{Y}_{std} \leftarrow$ Predict($\mathbf{X}_{UCB}, \hat{S}$)
4: $ucb \leftarrow \mathbf{Y}_{pred} + \beta \cdot \mathbf{Y}_{std}$
5: $\mathbf{X}_{new} \leftarrow$ argmax(ucb)
6:
7: **return** $\mathbf{X}_{new}$
8: **End Function**

The chosen acquisition function is upper-confidence bound (UCB), and its algorithm is shown in Algorithm 8. Initially, the surrogate model predicts the mean values $\mathbf{Y}_{\text{pred}}$ and their uncertainties $\mathbf{Y}_{\text{std}}$ for each sampled point $\mathbf{X}_{\text{UCB}}$. Then, the UCB value is computed for each sample using the formula from Eq. 3.22, where β is a parameter balancing exploration and exploitation. The next point is then selected as the maximum (or minimum, depending on the optimization problem) of the UCB values.

4 A Simple Problem of Convex Optimization

To evaluate the methods, an experiment on a straightforward problem was conducted. A convex system was modeled given by $f = w_1^2 + w_2^2$, where the convex nature implies that through optimization, variables w_1 and w_2 should ultimately converge towards 0 when minimizing the objective. To ensure a fair comparison between methods, testing was done using identical parameters. Initial means were set to $\mu_{w1} = 18$, $\mu_{w2} = 23$, with variances $\sigma_{w1,w1} = \sigma_{w2,w2} = 30$, and the number of iterations was fixed at 50. For CEMgmm, 3 components were utilized. The additional means for these two extra components were set to $\mu_{w1} = 14,35$ and $\mu_{w2} = 50,9$, with variances $\sigma_{w1,w1} = \sigma_{w2,w2} = 50$ and $\sigma_{w1,w1} = \sigma_{w2,w2} = 5$ for the second and third components, respectively.

In Fig. 4.1, the depicted values illustrate the optimal objective function results f for each iteration. For CEM techniques, these values are computed as the averages of the initial n_{elite} values, while in Bayesian Optimization, each iteration represents a single simulation yielding the best result directly. It is noticeable that Bayesian Optimization and CEMnormal exhibit a persistent error, failing to converge towards zero. In contrast, both CEMgmm and CEMsurrogate methods achieve zero error. However, CEMsurrogate converged in fewer iterations, thanks to its superior exploitation of elite samples. While CEMgmm and CEMnormal update their distributions using 10 elite samples, CEMsurrogate explores promising areas more extensively by generating 210 samples from them.

The runtime for each method is displayed in Table 4.1. The values represent the average results from five experiments each, along with their variances. It is evident that the Bayesian method has the longest execution time. This is due to the use of acquisition functions which causes the evaluation process to become more computationally intensive as the number of samples in their search space increases. Among the CEM

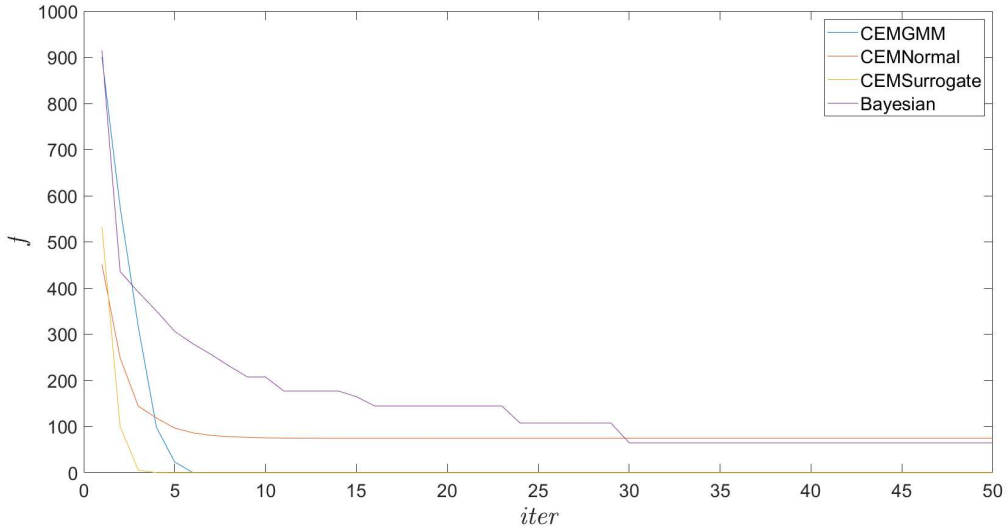


Figure 4.1: Best objective function values through iterations for each method

techniques, the Surrogate Model method exhibits the longest runtime due to its inherent complexity.

Table 4.1: Comparison of execution times of the methods

	CEMnormal	**CEMgmm**	**CEMsurrogate**	**Bayesian**
Time (s)	39.17 ± 3.18	40.32 ± 1.68	45.22 ± 1.20	125.68 ± 1.96

The drawback to Bayesian optimization is it introduces two new parameters that need to be tuned manually, which contradicts its overall purpose. The impact of modifying these parameters can be seen in Fig. 4.2. A larger β leads to more exploration of the search space which helps to avoid getting stuck in local optima, but also causes convergence to occur later in the iterations, and it increases the granularity of the graph. Conversely, increasing the number of samples used by the acquisition function reduces the final deviation from zero. The downside to n_{UCB} is runtime prolongation, making the execution time of the variant with 100 000 UCB samples 1.62e+03s – 13 times longer than the variant with 20 000 samples.

The surface plots of the objective function values and its parameters are illustrated in Fig. 4.3, 4.4, 4.5 and 4.6. Bayesian Optimization in Fig. 4.6 appears less stochastic compared to CEM, since it smooths out the optimization process over iterations, and this happens because it balances exploration and exploitation based on uncertainty. In contrast, CEM optimization relies more on direct sampling and iterative updates. CEM-

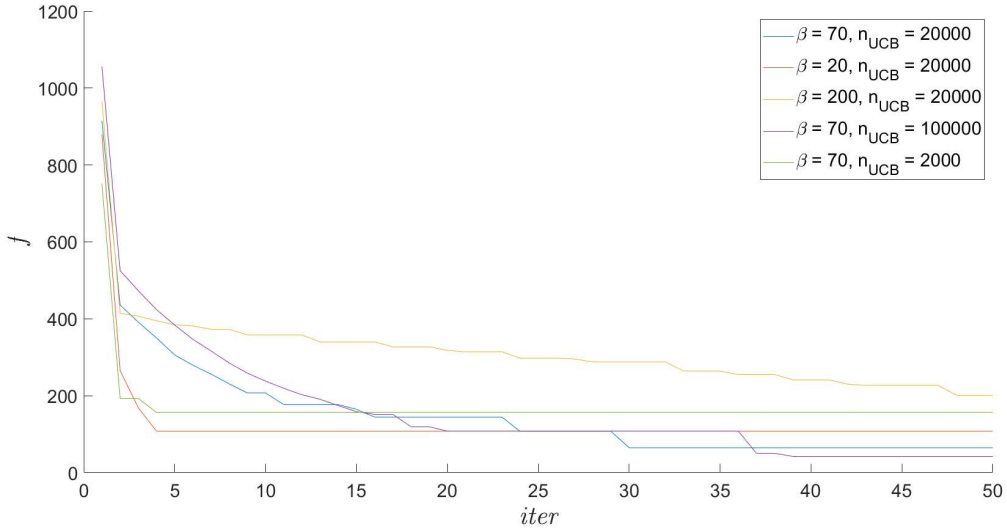


Figure 4.2: Comparison of different parameter combinations in Bayesian Optimization

surrogate also shows less randomness compared to other methods within its class.

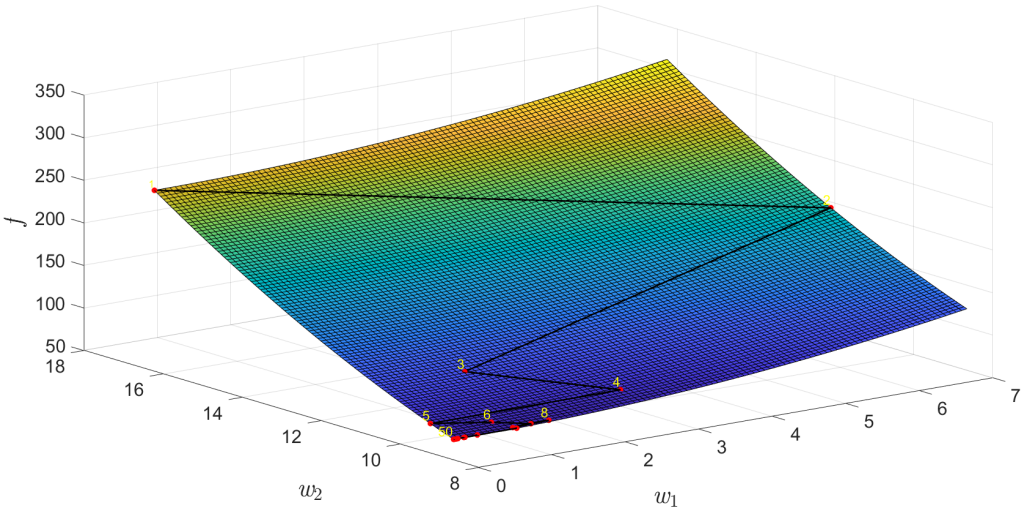


Figure 4.3: Surface plot of parameters and objective values for CEMnormal

For fitting testing, the progress of Gaussian Mixtures across multiple iterations was visualized in Fig. 4.7. The blue dots represent the samples, and the red dots highlight the elite samples. As the iterations proceed, it can be observed that the Gaussian components become more alike and their centers gradually shift towards zero. This reflects the iterative refinement and alignment of the mixture components, converging in the end towards a consensus central point.

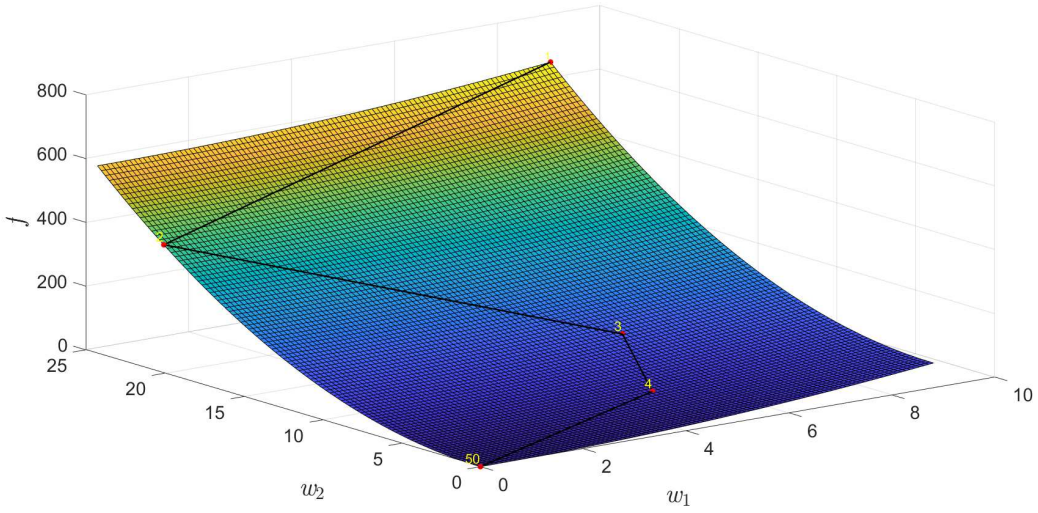


Figure 4.4: Surface plot of parameters and objective values for CEMgmm

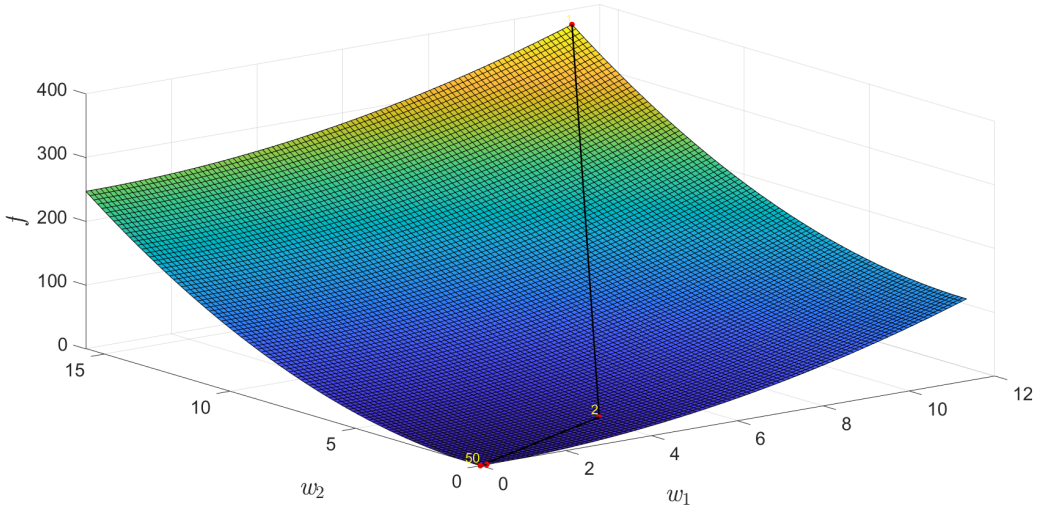


Figure 4.5: Surface plot of parameters and objective values for CEMsurrogate

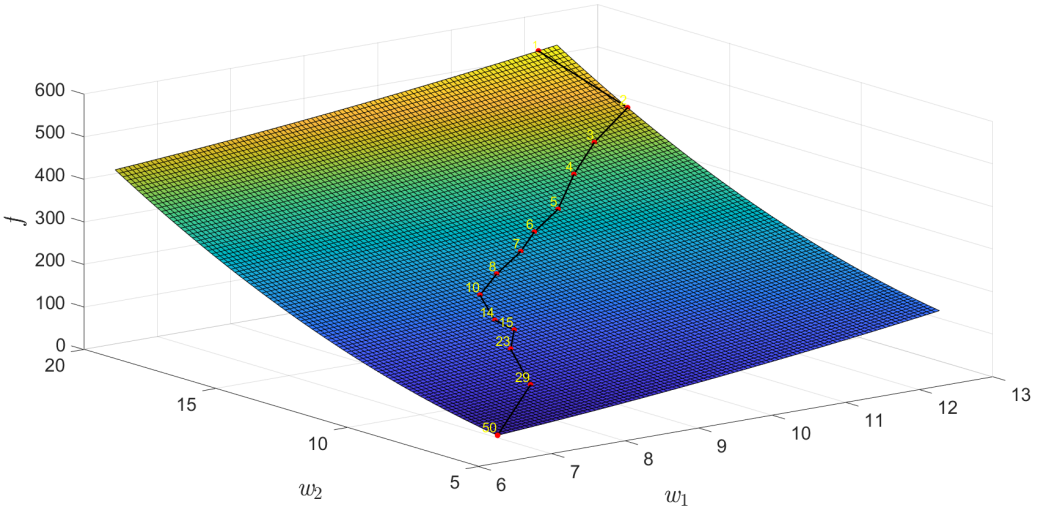
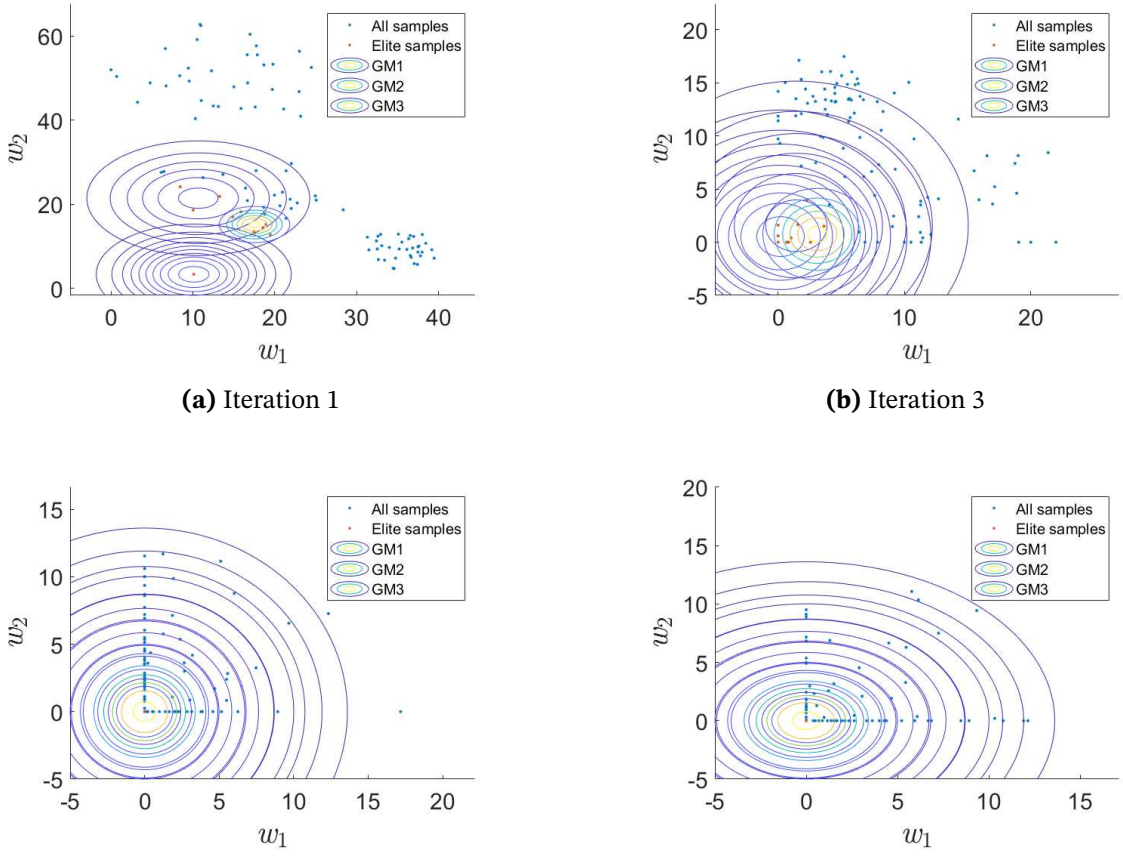


Figure 4.6: Surface plot of parameters and objective values for Bayesian optimization



(a) Iteration 1

(b) Iteration 3

(c) Iteration 5

(d) Iteration 20

Figure 4.7: Sampled points and contour lines of GMM Components

5 MPC Hyperparameter Optimization Results

The hyperparameters that were optimized for the MPC in [11] are the weights for state models (later referred to as OV_weights: s, n, μ and v_x) and weights for input models (later referred to as MV_weights: M and δ), as desribed in Section 2.2. This results in a total of nine variables to be optimized, making it a high-dimensional optimization problem.

5.1 Average Longitudinal Velocity as Objective Function

In the first case, the objective function chosen was to maximize the average longitudinal velocity on the track, represented by $f = \overline{v_x}$. Fig. 5.1, 5.2, 5.3, and 5.4 illustrate the progress of $\overline{v_x}$ over iterations across three sets of experiments for each method. More experiments were shown since the methods are stochastic and they result in different behaviours each time, depending on their starting points. In this high-dimensional problem, CEMnormal and CEMgmm demonstrated to be the most reliable methods. For the CEMgmm method, four components were utilized. This method required the fewest iterations to reach a steady state. Bayesian Optimization, while not always yielding optimal results, generally secured the best possible outcomes given the range of samples of its acquisition function. On the other hand, the CEM methods more occasionally became trapped in local optima, as observed in experiment 2 of CEMsurrogate. For this reason, it is recommended to repeat the optimization multiple times.

Table 5.1 summarizes the final best weights and their corresponding best average longitudinal velocity from the displayed experiments, for each method.

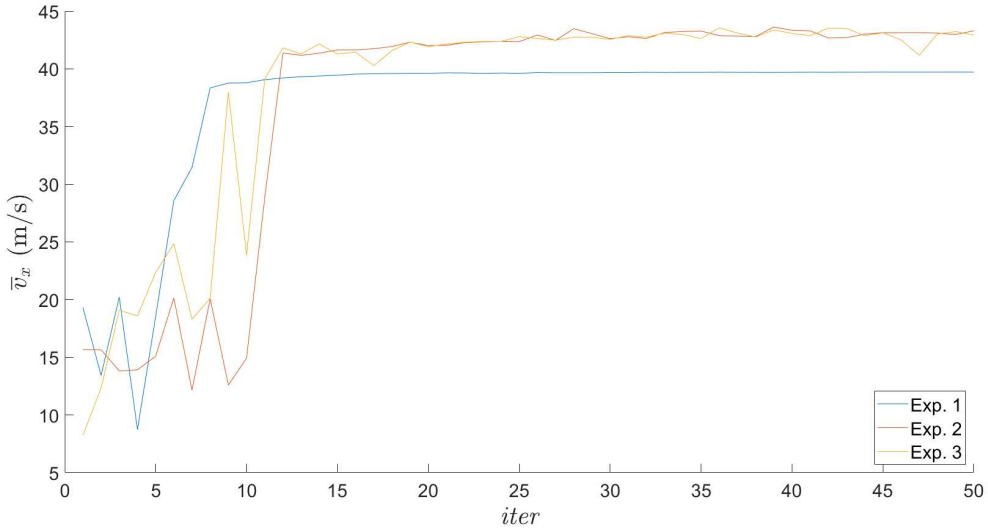


Figure 5.1: Best $\overline{v_x}$ values through iterations for CEMnormal

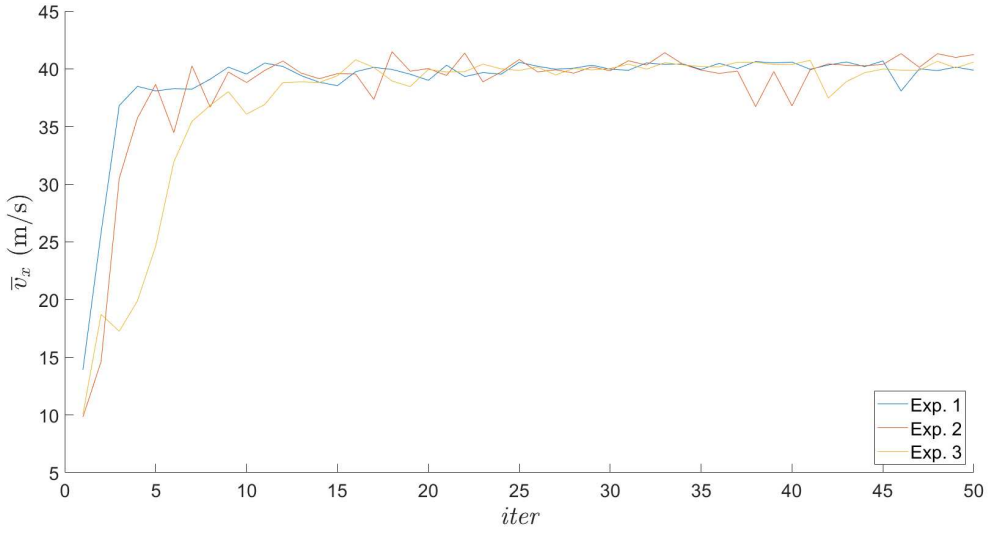


Figure 5.2: Best $\overline{v_x}$ values through iterations for CEMgmm

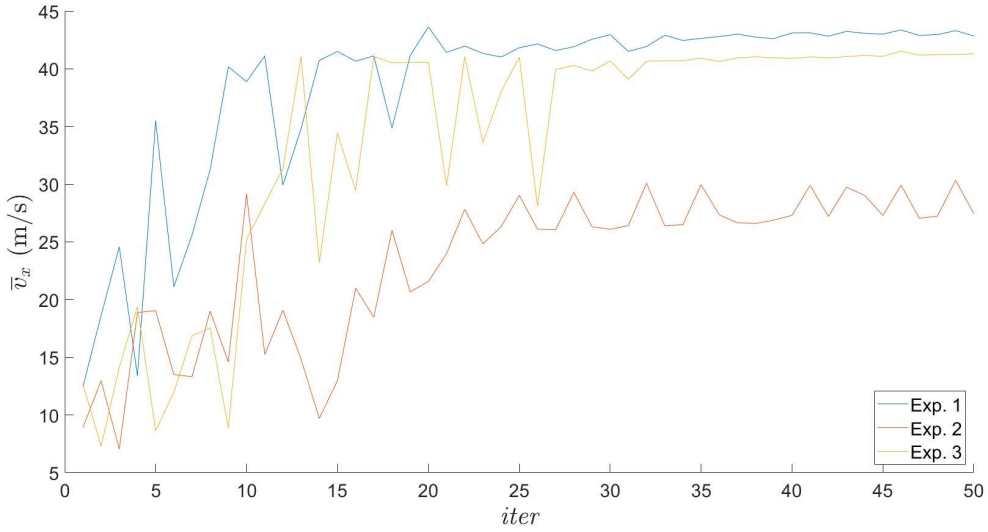


Figure 5.3: Best $\overline{v_x}$ values through iterations for CEMsurrogate

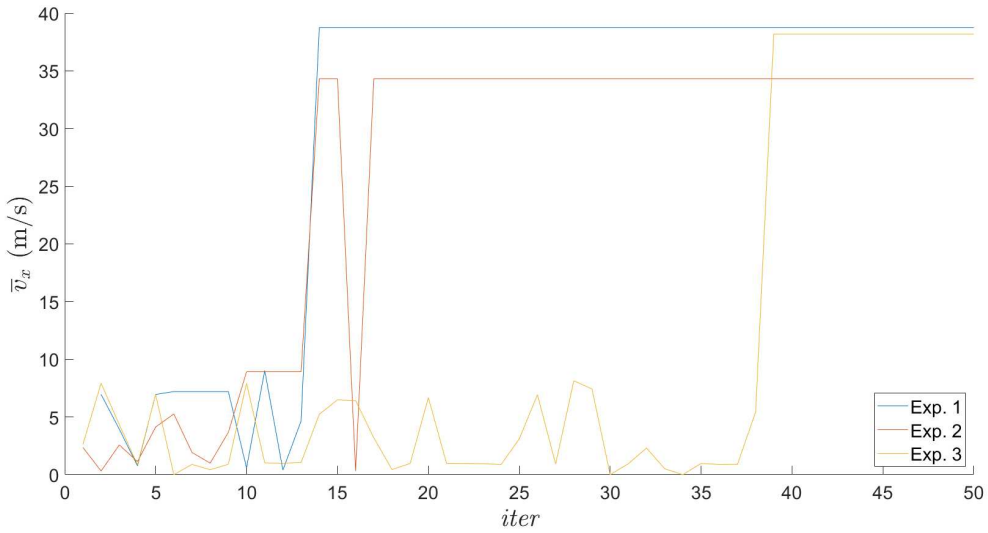


Figure 5.4: Best $\overline{v_x}$ values through iterations for Bayesian Optimization

Table 5.1: Final best weights and objective function values of each method

	CEMnormal	**CEMgmm**	**CEMsurrogate**	**Bayesian**
MV_weights	4.01	0.28	2.78	1.56
	7.72	5.86	1.78	0.10
	0.10	0.10	0.54	0.10
	0.62	0.89	0.11	3.22
	92.76	0.28	81.06	105.29
OV_weights	35.86	75.87	45.45	28.75
	51.81	26.05	52.50	47.68
	52.15	81.76	18.07	32.96
	61.93	72.08	54.68	42.47
$\overline{v_x}$ **(m/s)**	44.00	41.56	43.01	38.75

5.2 A More Appropriate Objective Function

Using only the average longitudinal velocity on a trajectory as the objective function is not an ideal choice, since the vehicle's velocity might increase at the expense of greater lateral displacement from the track centerline (n) and larger angular deviations from the track centerline (μ), or from the reference path. For instance, the vehicle in the simulation could just end up accelerating on a straight path instead of following the desired trajectory. Similarly, shortest simulation time duration is also not a good choice for the objective function. Therefore, to address this issue, the optimization problem was evaluated using a new objective function defined as:

$$f = \left|\overline{v_x}\right| - \left|\overline{n}\right| - \left|\overline{\mu}\right| \tag{5.1}$$

The aim is for the vehicle to achieve the highest possible velocity while minimizing both n and μ. For this MPC simulation, it was not necessary to use absolute values for v_x since the vehicle is configured to only move forward. The evaluation was conducted using the CEMnormal method, with the same means and variances as in the previous scenario. The final best weights and the corresponding average state values are presented in Table 5.2. Now, the final best value for $\overline{v_x}$ is lower than in the previous experiment, but the deviations of $\overline{\mu}$ and $\overline{n}$ are kept low.

The progression of the objective function and the best average state values over each iteration is illustrated in Fig. 5.5, 5.6, 5.7 and 5.8.

Table 5.2: Final best weights and average state values using CEMnormal

Parameter	Value
MV_weights	0.19, 0.18, 0.48, 0.40, 100.40
OV_weights	67.93, 65.99, 43.97, 39.01
$\overline{v_x}$ **(m/s)**	8.71
$\overline{\mu}$ **(rad)**	0.01
$\overline{n}$ **(m)**	0.14

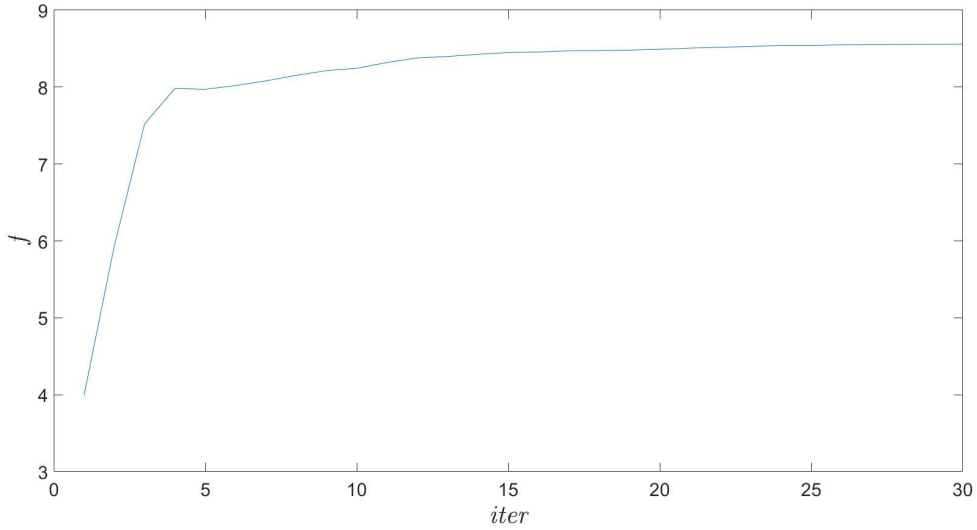


Figure 5.5: Best f values through iterations

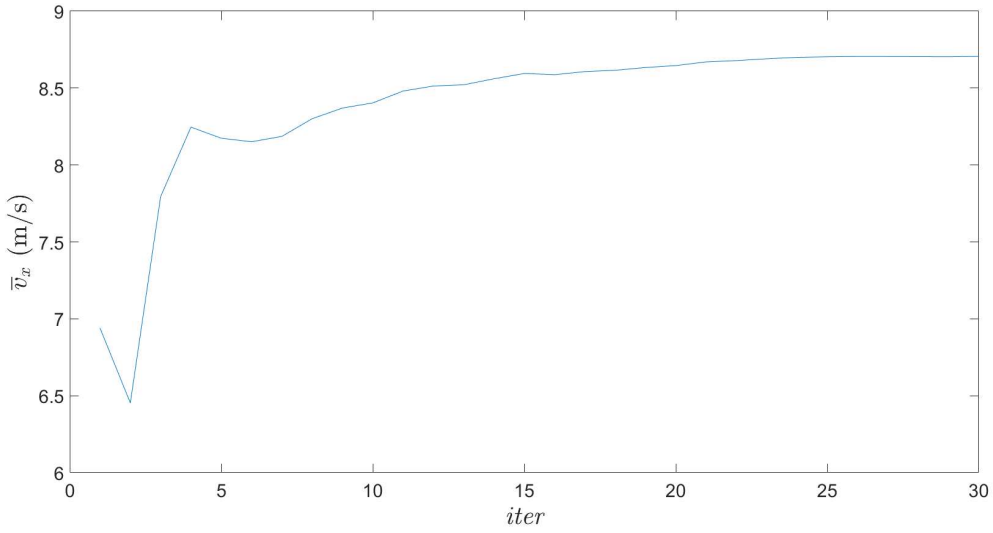


Figure 5.6: Best $\overline{v_x}$ values through iterations

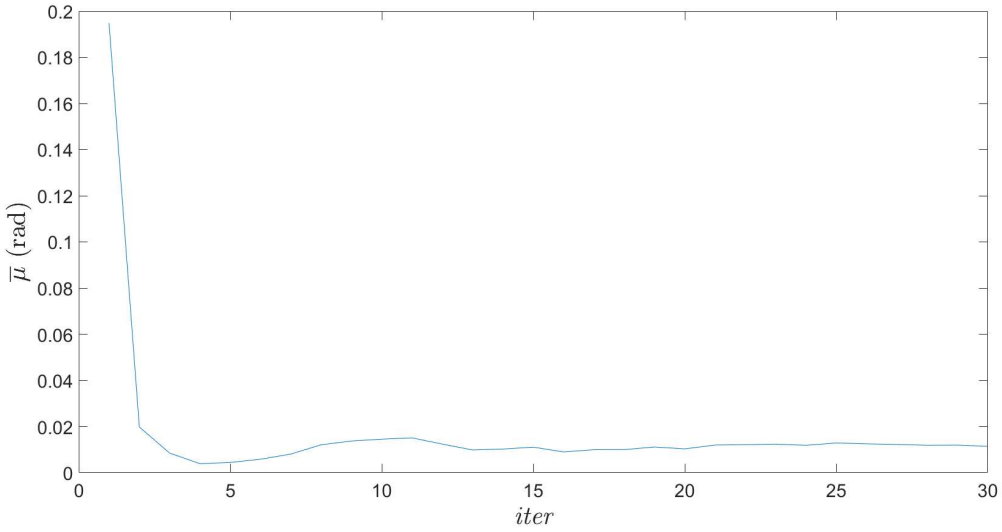


Figure 5.7: Best $\overline{\mu}$ values through iterations

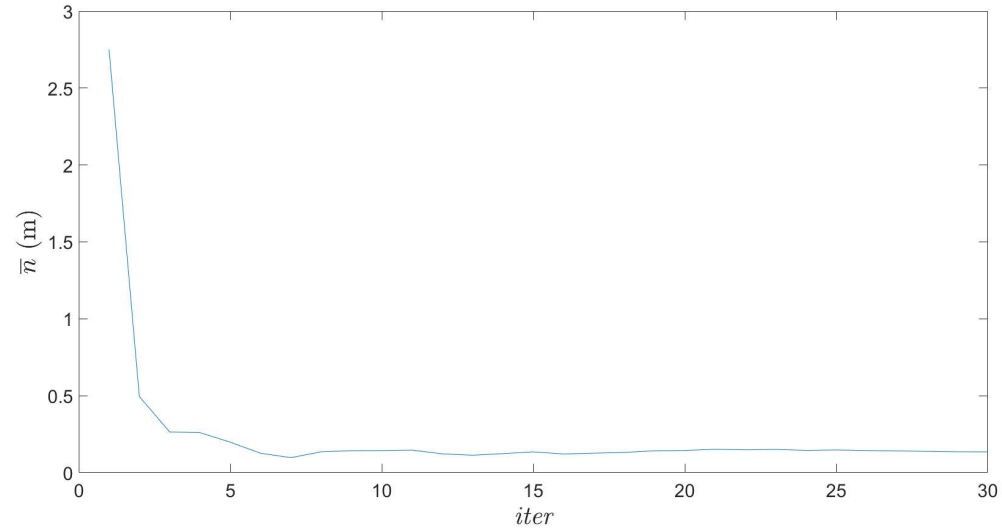


Figure 5.8: Best $\overline{n}$ values through iterations

6 Conclusion

Autonomous MPC hyperparameter optimization methods were successfully implemented, with an initial focus on maximizing average longitudinal velocity of a racing vehicle. The results demonstrated functionality and effectiveness of stochastic methods in high-dimensional spaces, for both Cross-Entropy Method and Bayesian Optimization. The CEM method with samples following a Gaussian Mixture distribution achieved optimal solutions with fewer iterations than the others. Bayesian optimization also showed to be a reliable technique for finding satisfactory solutions, but it introduces two additional parameters that require manual tuning. Despite this, a common issue arose across the methods due to occasional entrapment in local optima.

However, focusing solely on maximizing average velocity or minimizing simulation time as the objective function appears impractical in real-world scenarios. To address this, a more refined objective function was introduced, incorporating lateral displacement and angular deviation alongside velocity. As a result, although the highest average velocity achieved saw a decrease, there were significant improvements in reducing both lateral and angular deviations, consequently improving the precision of trajectory following.

To conclude, this study simplified the time-consuming and challenging task of MPC tuning, resulting in optimal outcomes and thereby enhancing overall vehicle performance.

References

[1] N. Morovat, A. K. Athienitis, J. A. Candanedo, and B. Delcroix, "Model-based control strategies to enhance energy flexibility in electrically heated school buildings," *Buildings*, vol. 12, no. 5, 2022. https://doi.org/10.3390/buildings12050581

[2] A.-R. Youssef, E. Mohamed, and A. Ali, "Model predictive control for grid-tie wind energy conversion system based pmsg," 02 2018. https://doi.org/10.1109/ITCE.2018.8316668

[3] P. Schwab, A. D. Schütte, B. Dietz, and S. Bauer, "Clinical predictive models for covid-19: Systematic study," 2020.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html

[5] N. Tsokanas, R. Pastorino, and B. Stojadinovic, "Adaptive model predictive control for actuation dynamics compensation in real-time hybrid simulation," *Mechanism and Machine Theory*, vol. 172, p. 104817, 03 2022. https://doi.org/10.1016/j.mechmachtheory.2022.104817

[6] E. Kayacan, W. Saeys, H. Ramon, C. Belta, and J. Peschel, "Experimental validation of linear and nonlinear mpc on an articulated unmanned ground vehicle," *IEEE/ASME Transactions on Mechatronics*, vol. PP, pp. 1–1, 07 2018. https://doi.org/10.1109/TMECH.2018.2854877

[7] R. G. Sanfelice, *Hybrid Model Predictive Control*. Cham: Springer International Publishing, 2021, pp. 930–939. https://doi.org/10.1007/978-3-030-44184-5_100051

[8] K. Pereida and A. P. Schoellig, “Adaptive model predictive control for high-accuracy trajectory tracking in changing conditions,” *CoRR*, vol. abs/1807.05290, 2018. [Online]. Available: http://arxiv.org/abs/1807.05290

[9] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*. A Clarendon Press Publication, 01 2002.

[10] H. Wei, “Modeling of human driver behavior via receding horizon and artificial neural network controllers,” 12 2013. https://doi.org/10.1109/CDC.2013.6760963

[11] D. Đurašinović, “Slijeđenje zadane putanje cestovnog vozila zasnovano na modelskom prediktivnom upravljanju,” Master’s thesis, University of Zagreb, Faculty of Electrical Engineering and Computing, 2023.

[12] A. Bemporad and M. Morari, “Model predictive control toolbox for matlab – user’s guide,” Natick, Massachusetts, United States, 2024, accessed: June 10, 2024. [Online]. Available: "http://www.mathworks.com/access/helpdesk/help/toolbox/mpc/"

[13] MATLAB. (2024) Adaptive mpc controller. Accessed: June 10, 2024. [Online]. Available: https://www.mathworks.com/help/mpc/ref/adaptivempccontroller.html

[14] ——. (2024) Simulate linear parameter-varying (lpv) systems. Accessed: June 10, 2024. [Online]. Available: https://www.mathworks.com/help/control/ref/lpvsystem.html

[15] J. Mohammadpour Velni and C. Scherer, *Control of Linear Parameter Varying Systems with Applications*. Springer Science and Business Media, 01 2012. https://doi.org/10.1007/978-1-4614-1833-7

[16] H. Hoos, “Stochastic local search : methods, models, applications,” *Dissertations in Artificial Intelligence*, vol. 215, 01 1998.

[17] M. Aibin and V. Fulber-Garcia. (2024, March) Deterministic and stochastic optimization methods, beeldung. Accessed: June 10, 2024. [Online]. Available: https://www.baeldung.com/cs/deterministic-stochastic-optimization

[18] R. J. Moss, “Cross-entropy method variants for optimization,” 2020.

[19] J. Šnajder, “Grupiranje ii, machine learning 1 lectures,” Faculty of Electrotechnical Engineering and Computing, Zagreb, 2023.

[20] M. el Bekri, O. Diouri, and C. Dalila, “A review of the particle swarm clustering method for intrusion detection in iot,” *Journal of Theoretical and Applied Information Technology*, 05 2022.

[21] F. Leclercq, “Bayesian optimization for likelihood-free cosmological inference,” *Physical Review D*, vol. 98, 09 2018. https://doi.org/10.1103/PhysRevD.98.063511

[22] I. Marković and I. Petrović, “Lecture 4: Linear estimation in static systems, estimation theory lectures,” Faculty of Electrotechnical Engineering and Computing, Zagreb, 2023.

[23] P. I. Frazier, “A tutorial on bayesian optimization,” 2018.

[24] Wikipedia, The Free Encyclopedia. (2024) Bayesian optimization. Accessed: June 15, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Bayesian_optimization

Abstract

Stochastic Hyperparameter Optimization for Model Predictive Control in Autonomous Racing

Gabriela Vitez

This master's thesis explores automatic hyperparameter optimization techniques for Model Predictive Control (MPC) in autonomous vehicle racing using stochastic optimization. Autonomous vehicles in racing require precise controllers and techniques to maintain optimal trajectories and stability. The high-dimensional parameter space makes manual tuning of MPC hyperparameters complex. The methods are implemented in MATLAB and Simulink. Application of Cross-Entropy Method is investigated with various distributions, including multivariate Gaussian and Gaussian mixtures, along with the use of surrogate models. Bayesian optimization is also studied as an alternative approach for hyperparameter fine-tuning. Initially, the goal was to maximize average longitudinal vehicle speed, and this demonstrated the effectiveness of all the methods. Later, the objective function was adjusted to maximize speed while simultaneously minimizing lateral displacement and angular deviation from the track centerline. This improved overall performance by maintaining high velocity and ensuring better trajectory following.

Keywords: Model Predictive Control, stochastic optimization, MATLAB, Simulink, Cross-Entropy Method, Bayesian Optimization, autonomous vehicles

Sažetak

Stohastičko optimiranje hiperparametara za modelsko prediktivno upravljanje u autonomnim utrkama

Gabriela Vitez

Ovaj diplomski rad istražuje automatske tehnike optimizacije hiperparametara modelskog prediktivnog upravljanja (MPC) za autonomne utrke vozila koristeći stohastičko optimiranje. Autonomna vozila u utrkama zahtijevaju precizne regulatore i tehnike za održavanje optimalnih trajektorija i stabilnosti. Visokodimenzionalnost prostora parametara čini ručno podešavanje hiperparametara u MPC-ima složenim. Metode su implementirane u MATLAB-u i Simulinku. Ispituje se primjena metode unakrsne entropije s različitim razdiobama, uključujući multivarijatnu Gaussovu i Gaussove mješavine, te uz korištenje surogatnih modela. Također se proučava Bayesovska optimizacija kao alternativni pristup za podešavanje hiperparametara. Početno je cilj bio maksimizirati prosječnu longitudinalnu brzinu vozila, što je pokazalo učinkovitost svih metoda. Kasnije je kriterijska funkcija prilagođena tako da maksimizira brzinu uz istovremeno minimiziranje lateralnog pomaka i kutnog odstupanja od središnje linije staze. To je rezultiralo poboljšanjem cjelokupnog ponašanja sustava održavajući visoku brzinu uz preciznije praćenje trajektorije.

Ključne riječi: modelsko prediktivno upravljanje, stohastička optimizacija, MATLAB, Simulink, metoda unakrsne entropije, Bayesova optimizacija, autonomna vozila